Is Dynamic Memory Management Dynamic Enough?

Jeff Keasler

High Speed Computing Conference

Salishan 2008

LLNL-PRES-403029

Prepared by LLNL under Contract DE-AC52-07NA27344

Lawrence Livermore National Laboratory

Unix/C Dynamic Memory Management

- Malloc() is used to allocate contiguous blocks of (virtual) heap memory.
- Malloc() API was invented for the PDP 7
  - 1.75us memory access time was *faster* than the 3.5us add instruction time.
  - Memory access had a uniform speed.
- Memory access time is now
  - 200+ times *slower* than the add instruction time.
  - Hierarchical in nature.

Unix/C Memory Management
Does It Need an Upgrade?

- The Malloc() API is a simple model that people are comfortable with.
- The Malloc()/C API could be extended to capture performance throughout the memory hierarchy.
- Is there evidence a change to the Malloc()/C API would be worth pursuing?

Hardware Optimizations Often Require Memory Alignment

BG/L memory throughput: a[i] = b[i] + ss*c[i]

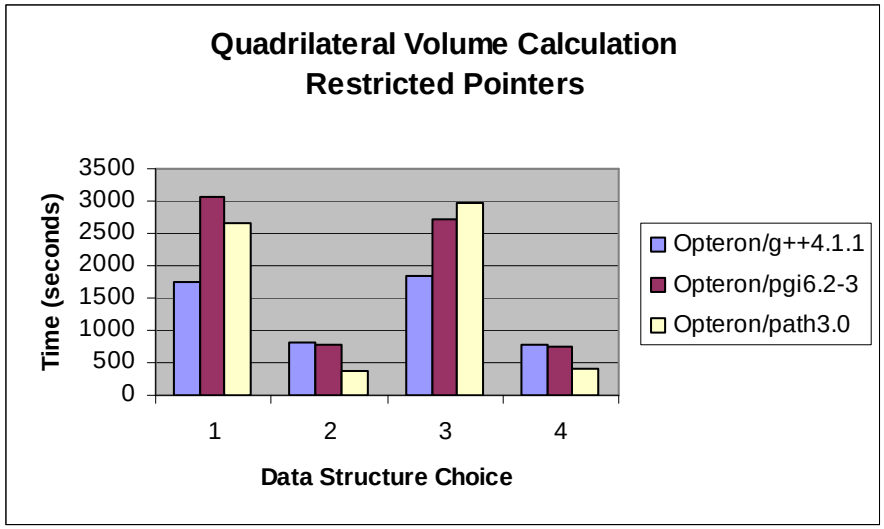
Array Size	Unaligned(MB/s)	Aligned(MB/s)
100	3040	6300
1000	3340	8270
10000	1290	3720
100000	1290	3720
500000	1290	1830
1000000	1280	1440

Results: Norris, Hartono, Gropp

Hardware Optimizations Depend on Memory Layout Choices

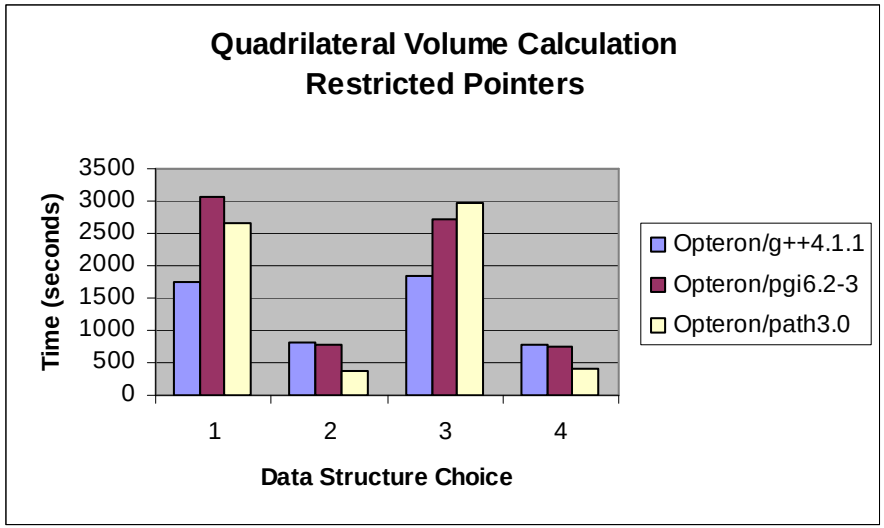
- An x86 SSE enabled processor often optimizes well with unaliased aligned array-like data:
  - ```
    double *x = new double[10000] ;
    double *y = new double[10000] ;
    double *z = new double[10000] ;
    ```
- But some applications/architectures optimize best with struct-like data due to reduced *register pressure* or better use of *prefetch streams*:
  - ```
    typedef struct { double x, y, z ; } Coord_t;
    Coord_t *coord = new Coord_t[10000] ;
    ```

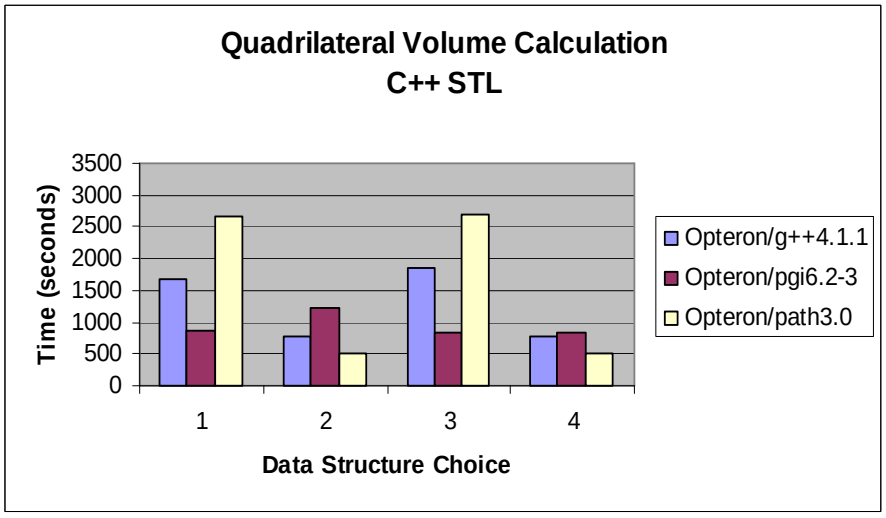
Compiler Optimizations Depend on Memory Layout Choices



GNU sees good optimizations for the first and third layout, while PathScale sees good optimizations for the second and fourth.

Compiler Optimizations Depend on Memory Layout Choices





GNU sees good optimizations for the first and third layout, while PathScale sees good optimizations for the second and fourth.

PGI sees more optimizations when using the STL.

Extending Malloc() Could Improve Reuse of Cache Memory

- Malloc()/C could be extended to minimize inter-array cache conflict
  - Modify base address and padding (Rivera, Tseng).
  - Separate declaration and allocation phase?

- Malloc()/C could be further extended to minimize inter-core cache conflict
  - Requires use of HugeTLB pages.
  - Would work best with O/S and hardware thread-team scheduling support.

Unix/C Dynamic Memory Management

- Effective dynamic memory management is key to performance portability across emerging architectures.
- Would it be acceptable to explore new models for Malloc()/C interactions?
- Could we reap the benefits without any changes to existing legacy code?